



下载APP



10 | 有哪些已经投入生产的 WebAssembly 真实案例？

2020-09-25 于航

WebAssembly入门课

[进入课程 >](#)**讲述：于航**

时长 15:56 大小 14.60M



你好，我是于航。

本节课，我们将不再“拘泥”于 Wasm 的实现细节，而是要从技术标准走向生产实践。作为应用篇中的第二节课，我们将一起来看看从 2017 年 Wasm MVP 标准的确定，直到如今 WASI 出现，使得 Wasm 走出 Web 的这几年时间里，现实世界中有哪些已经投入生产的 Wasm 真实案例？而这些案例又是怎样利用 Wasm，解决了哪方面实际问题的呢？

（这节课里介绍的几个案例，均由我总结于网络上相关公司发布的文章或视频分享。）

eBay - Barcode Scanner



第一个我们要介绍的实际案例来自于 eBay 在 Wasm 上的一次尝试。

eBay 是一家知名的线上拍卖与购物网站，人们可以通过 eBay 来在线出售自己的商品。作为一家知名的购物网站，为了优化用户录入待售商品的操作流程，eBay 在自家的 iOS 与 Android 原生应用中提供了“条形码扫描”功能。

通过这个功能，应用可以利用移动设备的摄像头扫描产品的 UPC 条形码，然后在后台数据库中查找是否有已经提交过的类似商品。若存在，则自动填写“商品录入清单”中与该物品相关的一些信息，从而简化用户流程，优化用户体验。

问题所在

在 iOS 与 Android 原生应用中，eBay 借助了自研的、使用 C++ 编写的条形码扫描库，来支持 UPC 条形码的扫描功能。而这对于诸如 iOS 与 Android 等 Native 平台来说，条形码的实际扫描性能得到了不错的保障，应用表现良好。

但是随着 eBay HTML5 应用的使用人数越来越多，为了能够使用户的商品录入流程与 Native 应用保持一致，“如何为 HTML5 应用添加高效的条形码扫描功能？”便成为了 eBay 工程师团队亟待解决的一个问题。

初期，技术团队使用了 GitHub 上的开源 JavaScript 版本条形码扫描器，来为 HTML5 应用支持 UPC 条形码的解析功能。但随着不断收到的用户反馈，团队发现 JavaScript 版本的条形码扫描器仅能够在 20% 的时间里表现良好，而在剩下 80% 的时间中，条形码的实际解析效率却不尽如人意，用户的每一次扫码过程都无法得到一致、流畅的用户体验。

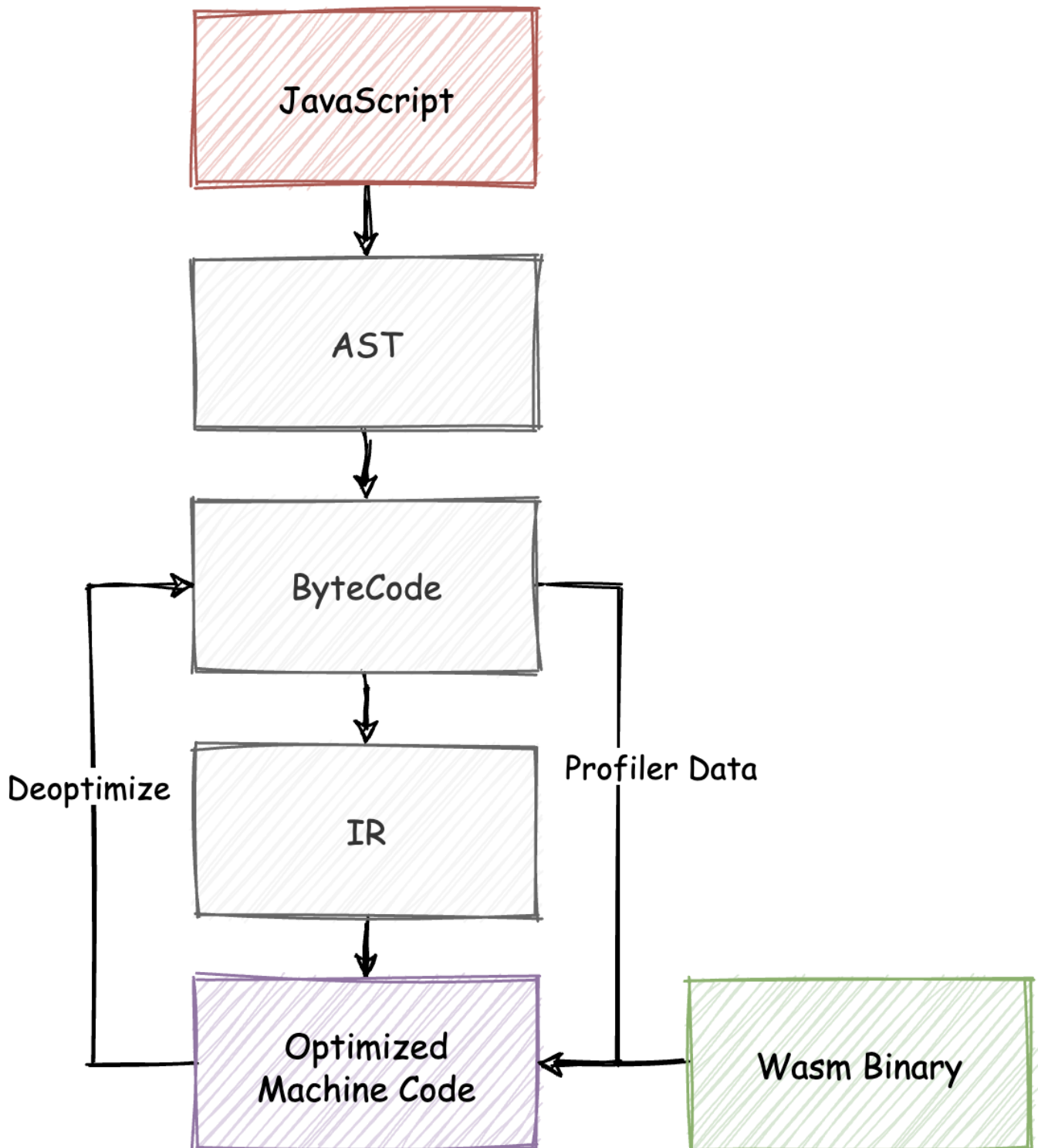
出现这种问题的一个最为重要的原因，便是由于 JavaScript 引擎在实际优化代码执行的过程中，无法确保用户的每一次扫描过程都能够得到 JIT 的优化。JavaScript 引擎采用的“启发式”代码执行和优化策略，通常会首先通过 Profiling 来判断出“热代码”的具体执行路径，然后再调用 JIT 引擎来优化这段代码。而实际上，究竟哪段代码能够被优化，谁也无从得知。

可能的解决方案

那么，如何解决这个问题？其中的一个选择是等待 WICG（Web Incubator Community Group，Web 孵化社区群组）曾提出的“Shape Detection API”提案。这个提案提出了一系列的 API，可以让 Web 平台应用直接利用硬件加速或者系统相关的资源，来支持如

人脸识别、条形码识别等功能。但该提案目前仍处于起步阶段，要实现跨浏览器的兼容性还有很多路要走。

eBay 技术团队所想到的另外一个方案，便是 Wasm。从下图所示的 V8 引擎编译管道中你可以看出。相较于 JavaScript 而言，浏览器引擎在执行 Wasm 字节码时不需要经过诸如“生成 AST”、“生成 Bytecode 字节码”、“生成 IR”以及“收集运行时信息”等多个步骤。JavaScript 引擎的优化编译器后端可以直接将 Wasm 字节码转换为经过优化的机器码，进而以接近 Native 代码的效率来执行。



不仅如此，Wasm 字节码在实际的执行过程中，也不会存在类似 JavaScript 代码的“去优化”过程，因此性能表现会更加稳定。

另一方面，借助于 Wasm 相关编译工具链的帮助，eBay 技术团队可以直接使用曾经为 Native 平台设计开发的 C++ 条形码扫描库。总的来说，eBay 技术团队不需要为 Wasm 重新编写这部分功能，而仅需要对已有的代码库进行少量改动即可。

项目架构

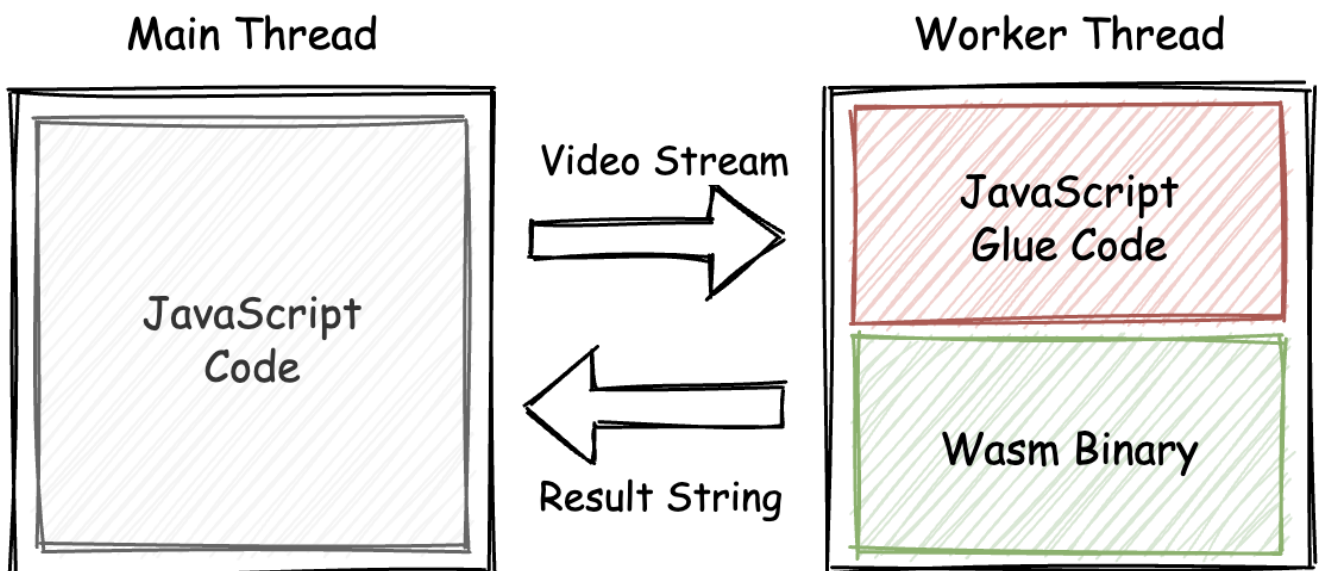
当方案确定之后，条形码扫描功能的具体工作流程如下所示。

使用 Web Worker API 从主线程创建一个工作线程（Worker Thread），用于通过 JavaScript 胶水代码来加载和实例化 Wasm 模块；

主线程将从摄像头获得到的视频流数据传递给工作线程，工作线程将会调用从 Wasm 模块实例中导出的特定函数，来处理这些视频流像素。函数在调用完成后，会返回识别出的 UPC 字符串或者返回空字符串，以表示没有检测到有效的条形码内容；

应用在运行时会通过设置“阈值时间”的方式，来检测是否读取到有效的条形码信息。当扫描时间超过这个阈值时，应用会弹出提示信息以让用户重试，或选择手动输入二维码序列。当然，阈值超时可能意味着两种情况：一种是用户没有扫描到有效的条形码；第二种是读取到的二维码视频流无法被应用使用的算法正确解析。

项目中使用到的 Wasm 模块以及 JavaScript 胶水代码，均是通过 Emscripten 工具链编译已有的 C++ 条形码扫描库得来的。整个方案的工作流程如下图所示。



一致化的编译管道

作为工程化的一部分，如何将 Wasm 模块的开发和编译流程，也一并整合到现有的 Web 前端项目开发流程中，是每个实际生产项目都需要考虑的事情。

一个 Wasm 模块，或者说是 Wasm Web 应用的完整开发流程涉及到多个部分。除了组成应用最基本的 HTML、CSS 以及 JavaScript 代码外，对于 Wasm 模块的开发和编译，我们还需使用到由 Rust 和 C++ 等系统级编程语言编写的模块源文件、相关的标准库，以及用于编译这些源代码的编译工具链，比如 Emscripten。

为了确保每次都能够在一个一致的环境中来编译和生成 Wasm 模块，同时简化整个项目中 Wasm 相关开发编译环境的部署流程。eBay 技术团队尝试采用了 Docker 来构建统一的 Wasm 编译管道。这样在每次编译 Wasm 模块时，Docker 都会启动一个具有相同环境的容器，来进行模块的编译流程，从而磨平了不同开发环境下可能带来的编译结果差异。

不仅如此，通过结合 NPM 下 “package.json” 文件中的自定义脚本命令，我们还可以让 Wasm 模块的开发与编译流程，与现有的 Web 前端应用开发编译流程，更加无缝地进行整合。举个例子，比如我们可以按照如下形式来组织 “package.json” 文件中的应用编译命令。

[复制代码](#)

```
1 {
2   "name": "my-wasm-app",
3   "scripts": {
4     "build:emscripten": "docker run --rm -v $(pwd)/src:/src trzeci/emscripten
5     "build:app": "webpack .",
6     "build": "npm run build:emscripten && npm run build:app",
7     // ...
8   },
9   // ...
10 }
```

其中，命令 “build:emscripten” 主要用于启动一个带有完整 Emscripten 工具链开发环境的 Docker 容器。并且在容器启动后，通过执行脚本 “./build.sh”，来编译当前目录下 “src” 文件夹内的源代码到对应的 Wasm 二进制模块。“build:app” 命令则用于编译原有 Web 应用的 JavaScript 代码。最后我们将两部分再进行整合，便得到了最终的 “build” 命令。

并不理想

以上基于 Wasm 的方案看起来十分理想。但经过实际测试后，eBay 技术团队发现，虽然基于 Wasm 的实现可以在 1 秒的时间内处理多达 50 帧的画面，但实际的识别成功率却只有 60%。剩下 40% 的失败情况大多是因为采样的画面角度不好，进而使得条形码的拍摄图像质量不高。产生问题的关键点，在于当前应用使用的是自研的 C++ 条形码扫描库。

自研的 C++ 条形码扫描库其一大特征为条形码的识别解析算法效率高，但仅适用于条形码成像质量较高的情况下。因此，急需一种方式来弥补在成像质量偏低时的条形码识别。

此时，团队将目光锁定到了另外一个业界十分有名的、基于 C 语言编写的开源条形码扫描库——ZBar。通过实验发现，当使用 ZBar 作为条形码扫描库时，在所设置的阈值时间范围内，整个应用的扫描成功率提高到了 80%。

但 80% 的成功率对于产品的用户体验来说仍然不够。团队继续对 ZBar 和自研的 C++ 条形码扫描库进行测试。在经过一段时间后，他们发现在某些 ZBar 超时的情况下，自研的 C++ 库却能够快速得到扫描结果。显然，基于不同的条形码图像质量，这两个库的执行情况有所不同。

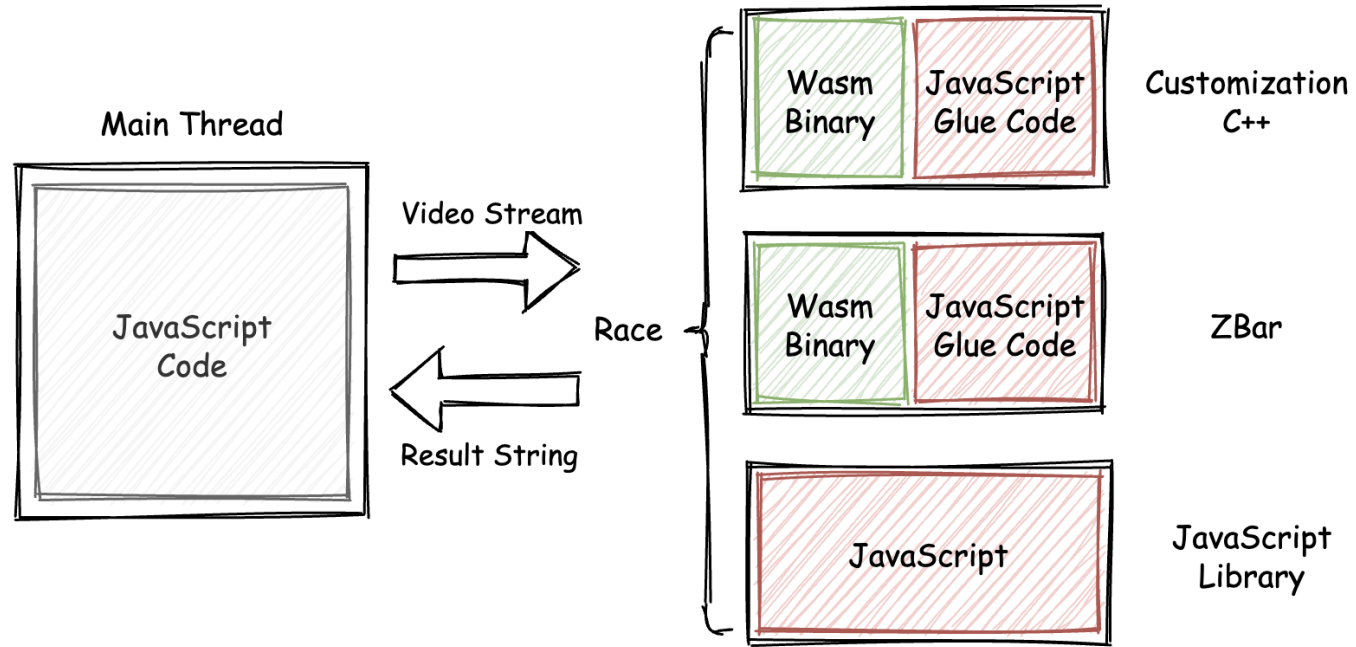
竞争取胜

为了能够同时利用 ZBar 和自研的 C++ 库，eBay 技术团队选择了一个“特殊的方案”。我想你肯定也能够猜到方案的大致内容。

在这个方案中，应用会启动两个工作线程，一个用于 ZBar，另一个用于自研的 C++ 库，两者同时对接收到的视频流进行处理。当主线程接收到有效的识别结果时，便结束所有工作线程的执行。若超时，则显示错误信息。

经过测试，条形码在不同模拟测试场景中的识别成功率，可以提高到 95%。

无独有偶的是，当尝试把 JavaScript 版本的条形码扫描器实现同样作为工作线程，加入到竞争“队列”中时，整个应用的条形码扫描识别成功率达到了将近 100%。这样的结果让人感到惊喜。应用的最终架构可以通过下图很好地进行展示。



产品上线后的最终效果如下图所示。



图片来源于 eBay 官方博客

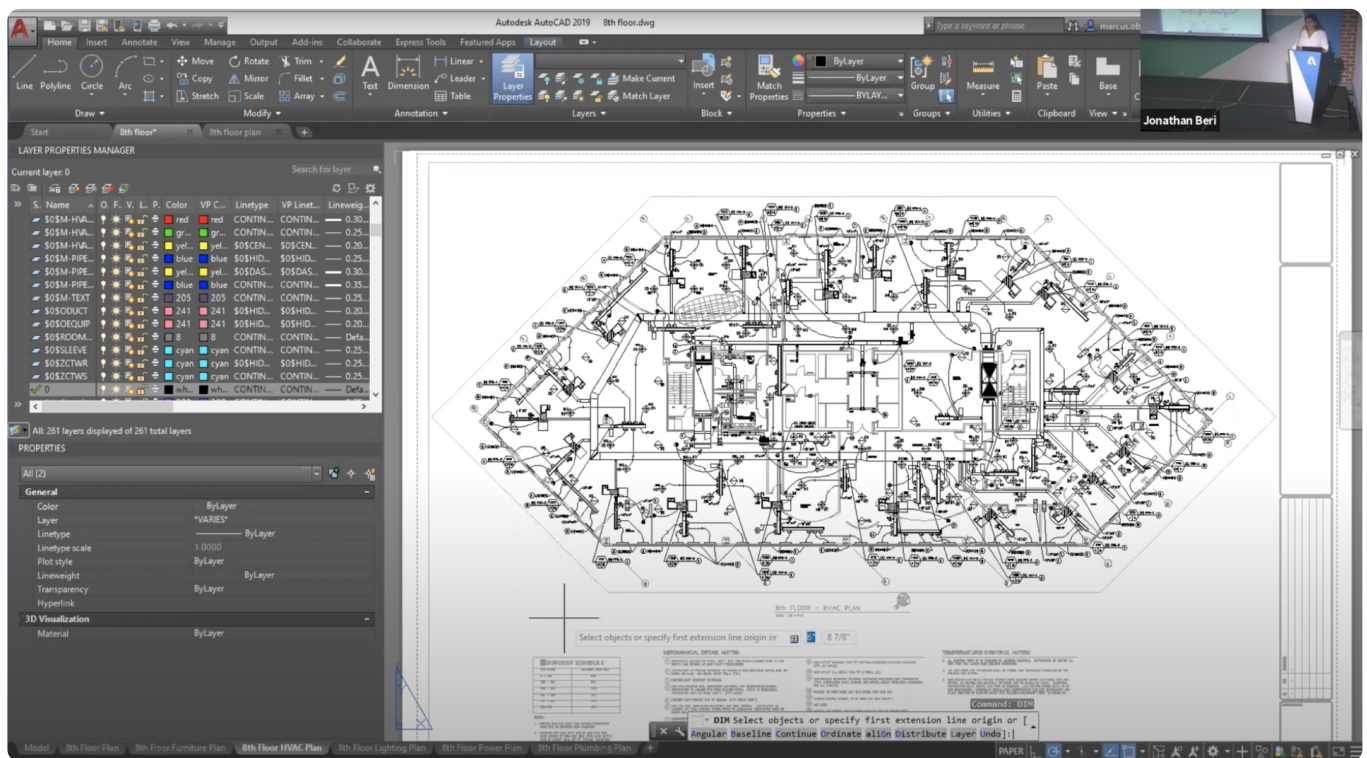
产品在线上使用了一段时间后，eBay 技术团队对应用的条形码扫描情况进行了统计，结果发现有 53% 的成功扫描来自于 ZBar；34% 来自于自研的 C++ 库。剩下的 13% 则来自于第三方的 JavaScript 库实现。可见，其中通过 Wasm 实现（自研 C++ 库、Zbar）得到的扫描结果占据了总成功次数的 87%。

虽然文章中没有提及，但实际上，设备对 Wasm 的兼容性也是需要考量的一个因素。你可以思考一下，我们怎样做可以在上述方案的基础上，来同时兼容旧设备上的条码扫描功能。

AutoCAD Web

第二个我们要介绍的案例来自于一个有着将近 40 年历史的知名设计软件 —— AutoCAD。

AutoCAD 是一款由 Autodesk 公司设计研发的，用于进行 2D 绘图设计的应用软件，它被广泛地用于土木建筑、装饰装潢、工业制图等多个领域中。相信大部分的工科同学，也一定在大学本科期间参与过 AutoCAD 的课程与相关考试。如下图所示，是该应用桌面端版本的运行截图。



图片来源于 YouTube

历史负担

AutoCAD 桌面端软件的发展有着将近 40 年的历史。而随着应用的不断发展，随之而来便是逐渐变大的代码库体积，以及不断复杂化的软件架构。截止 2018 年，AutoCAD 桌面端应用已经有着超过 1500 万行的 C/C++ 代码，并且仍然在以肉眼可见的速度增长着。

移动互联网浪潮

随着 2008 年移动互联网浪潮的逐渐兴起，越来越多的用户开始使用 PC 甚至是移动设备上的 Web 浏览器，来完成日常工作的一部分内容。感知到趋势的 Autodesk，便开始着手将自家的 AutoCAD 应用从 PC 端的原生应用逐渐向 Web 应用进行移植。

初期，由于 AutoCAD 原生应用本身的代码库过于庞大，AutoCAD 团队决定从头开始编写 AutoCAD 的 Web 版应用。在当时那个年代，HTML5 刚刚标准化，浏览器在功能特性上的支持还不够全面，并且跨浏览器的兼容性也很难得到保障。因此，AutoCAD 移植 Web 应用的第一版本便是基于 Adobe Flash 重新编写的，这个应用发布于 2010 年。

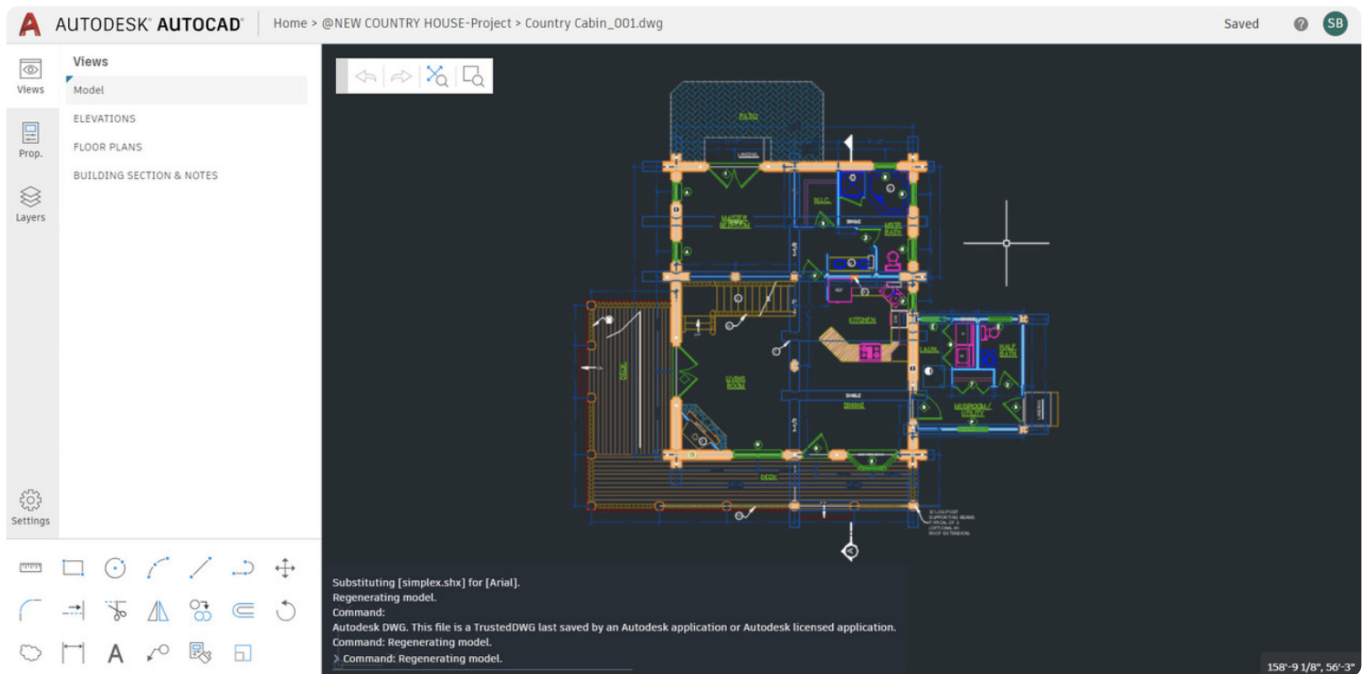
为了能够进一步利用 Web 标准，来优化 AutoCAD Web 应用的性能，并使得整个 Web 应用的技术架构更加贴近基于 JavaScript 构建的 Web 应用标准，AutoCAD 团队于 2013 开始着手进行 AutoCAD 标准 Web 应用的移植工作。并且此时的 AutoCAD 团队还有着更大的“野心”。

他们首先基于 C++，重写了为 iOS 移动端 Native 应用准备的轻量版代码库。然后通过交叉编译 (Tangible) 的方式，将这些 C++ 代码编译为了 Java 代码供 Android 设备使用。最后，在 Google Web Toolkit (一个 Google 开发的可以使用 Java 语言开发 Web 应用的工具集) 的帮助下，又将这些 Java 代码转译为了 Web 平台可用的 JavaScript 代码。

但事实上，由于 GWT 本身作为转译工具，会产生很多额外的胶水代码，并且经由 C++ 交叉编译而来的 Java 代码本身质量也并不高，因此这导致了最后生成的 Web 应用代码库十分庞大，且在浏览器中的运行性能并不可观。这个“粗糙版”的 Web 应用发布于 2014 年。

时间来到 2015 年，彼时 ASM.js 作为 Wasm 的“前辈”正展露着头角。AutoCAD 团队借此机会，在 Emscripten 工具链的帮助下，直接从 AutoCAD PC 版原有的 C++ 代码库中移植了一部分主要功能到 Web 平台上，ASM.js 所带来的性能提升，让团队对 AutoCAD Web 应用的进一步发展充满了期待。

2018 年 3 月，基于 Wasm 构建的 AutoCAD Web 应用诞生。

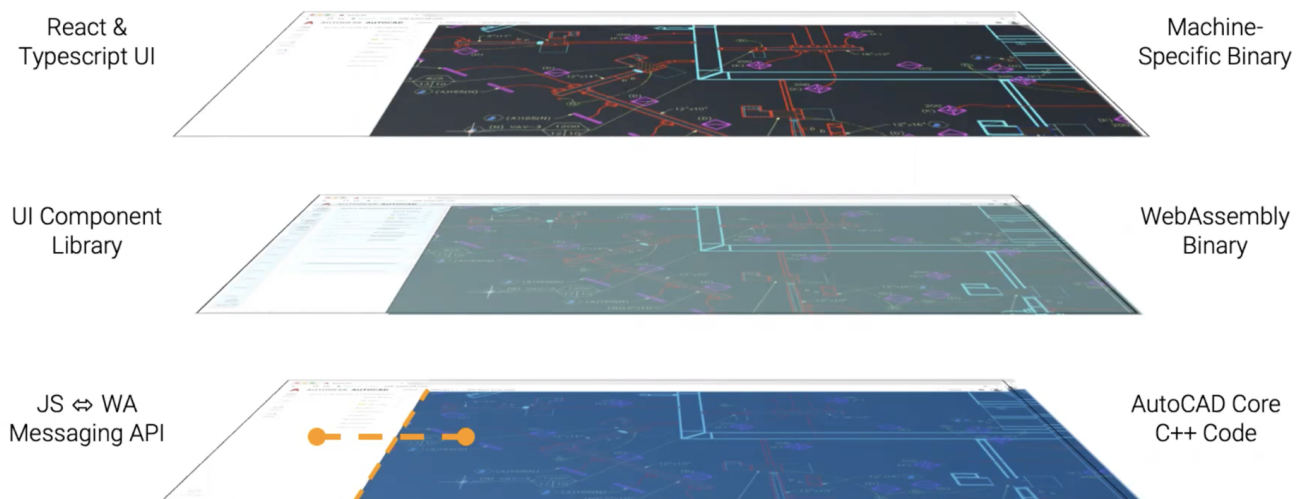


图片来源于网络

应用架构

整个 AutoCAD Web 应用的组成结构你可以参考下面这张图。在应用的右侧是绘图区域，该区域由 HTML 中的 Canvas 元素与相关 Web API 进行渲染，运行在独立工作线程中的 Wasm 模块实例则负责控制这部分区域的实际绘图效果。

左侧的 UI 控制区域由 TypeScript 基于 React 框架进行构建，基于组件化的构建方式与我们日常开发的 Web 前端应用项目基本保持一致。UI 部分的交互操作则会通过 “postMessage” 等 Web API 通知到工作线程中的 Wasm 实例，并对输出到 Canvas 中的画面进行实时处理。



图片来源于 YouTube

总结

好了，讲到这，今天的内容也就基本结束了。最后我来给你总结一下。

在这节课里，我们举了两个比较有代表性、在现实生活中的 Wasm 生产实践案例。第一个是 eBay 在其 Web H5 应用中添加的条形码扫描功能。eBay 技术团队在初期使用了第三方的 JavaScript 版本条形码识别库，来进行条形码的识别，但无奈识别成功率较低。

而随着后期 ASM.js 与 Wasm 的出现和普及，eBay 技术团队选择将自研的，原先被应用于 Native 平台的 C++ 识别库编译到 Wasm，并整合到 Web 应用中使用。此时虽然识别成功率有所上升，但在某些成像质量较差的场景下，条形码仍然无法被正确识别。

为了解决这个问题，团队成员又以同样的方式，将基于 C 语言开发的知名第三方条形码识别库 ZBar 编译到了 Wasm。并通过多个工作线程“竞争”的方式，尝试同时整合 JavaScript 版本实现、ZBar 与自研的 C++ 识别库，让应用的整体识别成功率有了一个质的提高。

在第二个案例中，我们介绍了 AutoCAD 在移动互联网浪潮兴起的这十年时间里，不断尝试将其 Native 应用移植到 Web 平台所使用的一些方式。而在这些众多的方案中，基于 Wasm 的方案给予了 AutoCAD 能够在 Web 平台上流程运行的可能。

最后，希望这些真实的案例能够给予你对 Wasm 更多的信心和思考。

课后练习

最后，我们来做一个思考题吧。

你觉得将 Native 应用移植到 Web 应用时可能会存在哪些问题呢？或者说 Native 应用与 Web 应用在执行流程或组成方式上有哪些区别呢？欢迎大家各抒己见。

今天的课程就结束了，希望可以帮助到你，也希望你在下方的留言区和我参与讨论，同时欢迎你把这节课分享给你的朋友或者同事，一起交流一下。

[提建议](#)

极客时间 3 周年

做任务 得千元礼包

[【点击】图片, 立即参加 >>>](#)

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 [09 | WebAssembly 能够为 Web 前端框架赋能吗？](#)

下一篇 [11 | WebAssembly 在物联网、多媒体与云技术方面有哪些创新实践？](#)

精选留言 (4)

[写留言](#)**pedro**

2020-09-25

可以总结一下，webassembly 非常适合迁移有其它语言沉淀的类库到 web 平台

作者回复: 没错的，这是在应用 Wasm 时可以纳入考虑的一个比较重要的特征。



2





慌慌张张

我们公司直播页面也是基于wasm的，感觉native和web上主要是编译库，好多特性在wasm不支持。



Geek_79ab1c

2020-10-09

这个画图的软件是啥

展开 ∨

作者回复: 工具在这边: <https://app.diagrams.net/>



一步

2020-09-26

现在国内有哪些知名的团队在使用 WebAssembly 技术，国内的案例？去招聘网站看了一下，几乎没有对应的职位的

展开 ∨

作者回复: 国内有一些公司在尝试性使用，但目前相对来说还是比较少的。比如华为、阿里云有做过调研。而全民 K 歌应该是有在生产环境中使用。目前应该还没有专门为 Wasm 设立的岗位，大多还是在观望。

